Enabling High Job Throughput for Uncertainty Quantification on BG/Q

SCICOMP 2014

May 27-30, 2014

John Gyllenhaal, Todd Gamblin,
Adam Bertsch, and Roy Musselman (presenter)

Lawrence Livermore National Laboratory

LLNL-PRES-654901

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Abstract

Traditionally, high-end supercomputers have been engineered for single, massively parallel grand challenge calculations. Recently, demand has grown for large ensembles of small-scale problems as part of parameter sensitivity studies. In the DOE, this work is mainly in the area of Uncertainty Quantification (UQ). UQ users of Sequoia, LLNL's IBM Blue Gene/Q system, need to run a massive number of 8-node, 8-process, or even single-process jobs. This stresses large machines in ways not originally intended. In particular, the management system is forced to track many small job allocations. Current resource managers run a process on the management node to track each active parallel job. On a system of Sequoia's size, (1.5 million cores), this can easily require tens of thousands of manager processes and exceed the limits of the Linux OS running on the front-end.

In this presentation, we describe techniques that allow UQ users to efficiently utilize all 96K Sequoia nodes. We describe hardware and software challenges we have overcome to allow resource managers to run more front-end processes. We also describe a new tool called "Cram", which allows many concurrent and independent MPI jobs to run within a single MPI job. Together, these approaches have allowed us to successfully run millions of concurrent, small MPI jobs on the Sequoia machine, without placing undue burden on its service nodes.

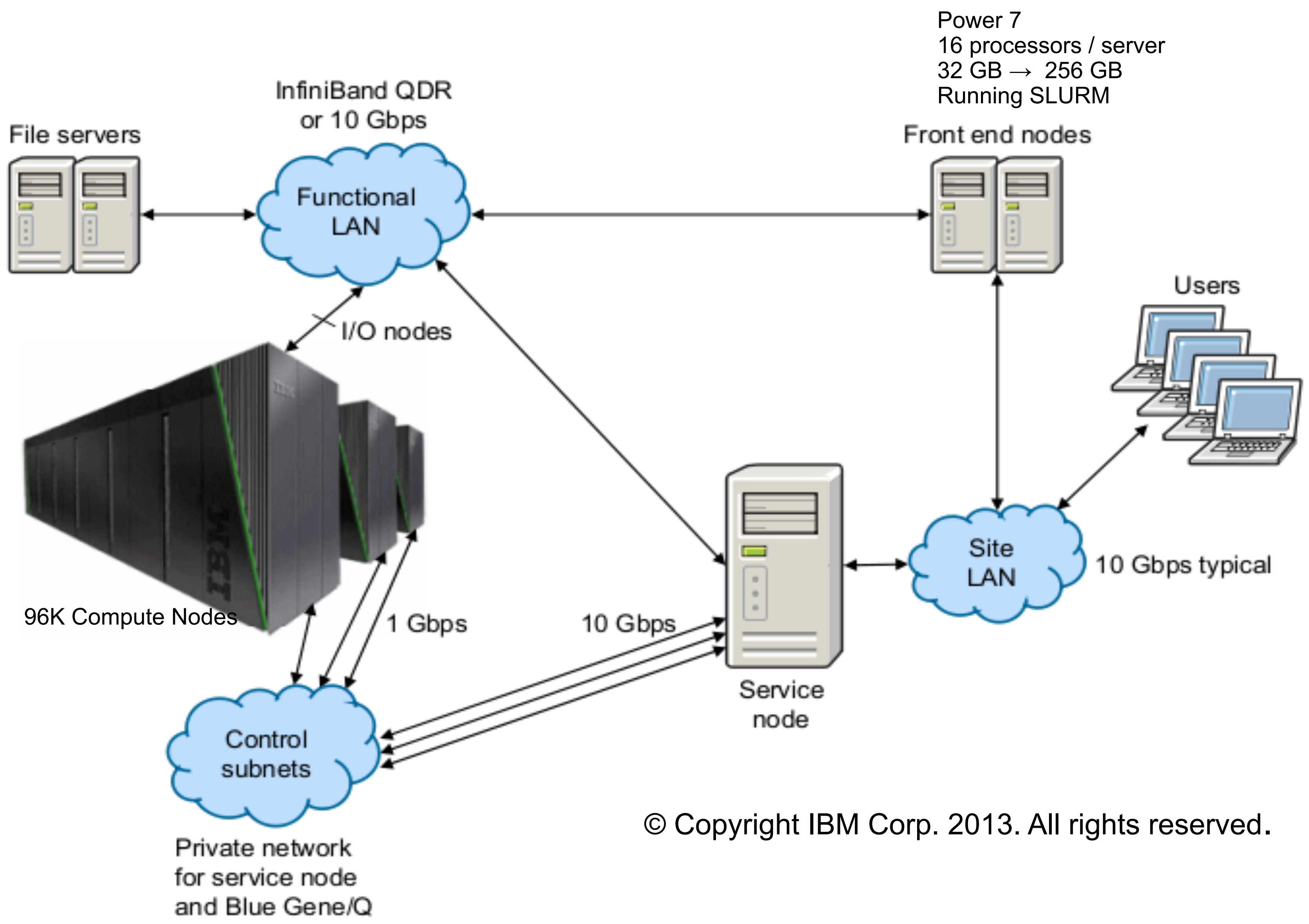
Size requirements of production jobs on Sequoia have changed

- LLNL's Sequoia is the largest BG/Q system
  - 96 racks
  - 192 midplanes
  - 6,291,456 MPI tasks at 64 tasks/node
- Procurement, SOW and acceptance testing focused on large jobs
  - — Tested 192 simultaneous 512 node jobs to one 96k node job
- Code regression tests start after codes ported
  - — Want to run many times daily ~1k small MPI jobs (mostly 1 or 4 mpi tasks)
  - — Although can't pack jobs effectively, at least can run on 1k nodes!

First signs of trouble at 200 simultaneous runs:

```
2013-05-02 11:58:21.239 (ERROR) [0x40001fca1e0]
39930:ibm.runjob.client.MuxConnection: could not write INSERT JOB
message: Transport endpoint is not connected
```

BlueGene/Q System Architecture



Limits Encountered During Scale Up

1. **Socket Connection Limit**
   - BGQ Efix 028 created to allow use of actual SOMAXCONN setting
   - Setting SOMAXCONN to 8192 eliminates: Transport endpoint is not connected
2. **User process limit**
   - Immediately hit user limits, 256 runs consumed all 1024 user processes
   - Raised descriptors and maxproc limits to 16384, should allow 4k simultaneous runs
3. **I/O Timeout limit**

```
A CIOS jobctl daemon running on I/O node R03-ID-J07 failed to
acknowledge a heartbeat.   Service Action: Reboot the I/O node.
```

   - At 512 simultaneous runs, I/O nodes 'crashed' and needed rebooting:
   - Temporary fix: Doubled I/O heartbeat to 120 seconds allowed 1k jobs
   - Real fix released in V1R2M1, we put I/O heartbeat back to 60 seconds
4. **SLURM limits**
   - After fix, new limit appears to be 3k simultaneous SLURM jobs before 'crash'
   - Experiments w/o SLURM using underlying runjob interface allow 16K jobs (8K per FEN).

Uncertainty Quantification - UQ

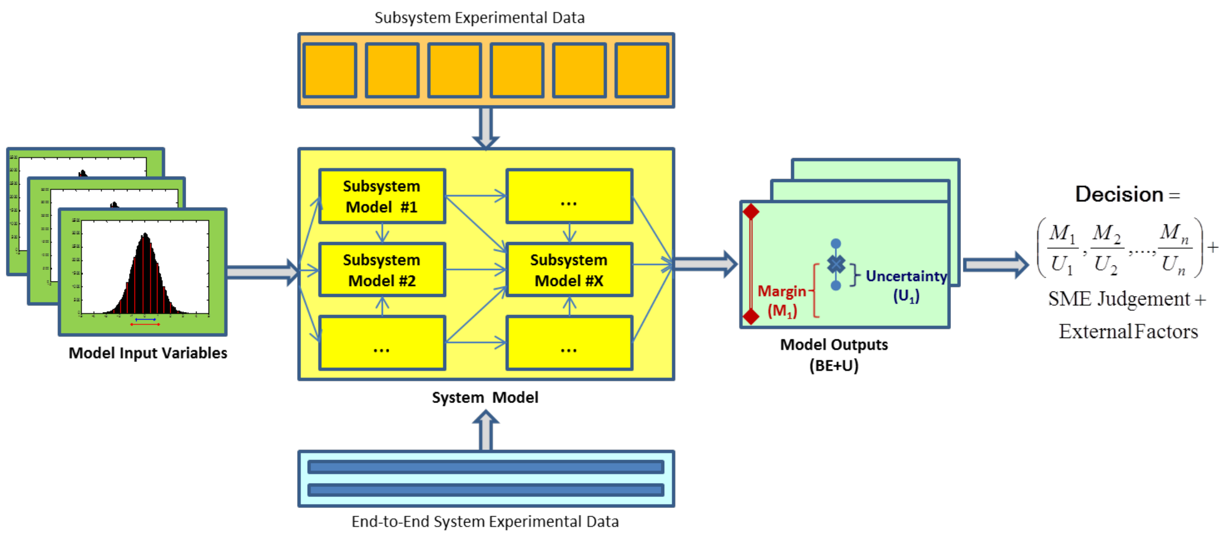
- Many computer simulations of natural sciences use models that inherently have many sources of uncertainty that can change the behavior of the model over time.
  - Examples: Input parameters, algorithm approximation inaccuracies, experimental measurements, interpolation errors.

- The application of UQ techniques help determine how likely certain outcomes are if some aspects of the system are not exactly known.

- At the most basic level, UQ consists of a series of simulations varying specific inputs in order to generate the error bars on the outputs with a given level of statistical uncertainty.

- Increasing the quantity of simulation results improves the accuracy of the uncertainty values. This translates to a simulation throughput challenge.
- Additional Information:
  - http://en.wikipedia.org/wiki/Quantification_of_margins_and_uncertainties

Uncertainty Quantification Users Jump In



- Several UQ groups get big allocations on Sequoia
  - First group ready planned to run 10k jobs at a time!
    - — We ask them to limit themselves to 1.5k jobs at start (3k limit then)
    - — They mention possible future desire for more, possibly 1 million runs!
  - Two days into their 1.5k run test, Sequoia crashes
    - — Cascading memory exhaustion on all batch nodes, all jobs lost
      - Most of their jobs happened to land on same batch node, OOM killed
      - After node dies, jobs moved to next batch node, OOM killed, repeat

Hard to Run 10k of Anything

- Need a lot of memory (exacerbated by 64k pages)
  - Srun 15MB x 10000 = 150GB of memory just to launch
  - Python 50MB x 10000 = 500GB! (eliminated via rewrite)
  - Increase FEN memory
    - 32GB -> 80GB (borrowed)-> 256GB (purchased)
  - Switched to 4k pages in kernel
    - 40% less memory used by srun, 3% slower compiles
    - Had to patch firefox, 64k pages hardcoded in

- Need a lot of pids (32k kernel max by default)
  - Srun uses 4 pids (threads) x 10k = 40k
  - For large runs, boost kernel and user limit to 64k

UQ Runs Exposes ‘Rare’ issues

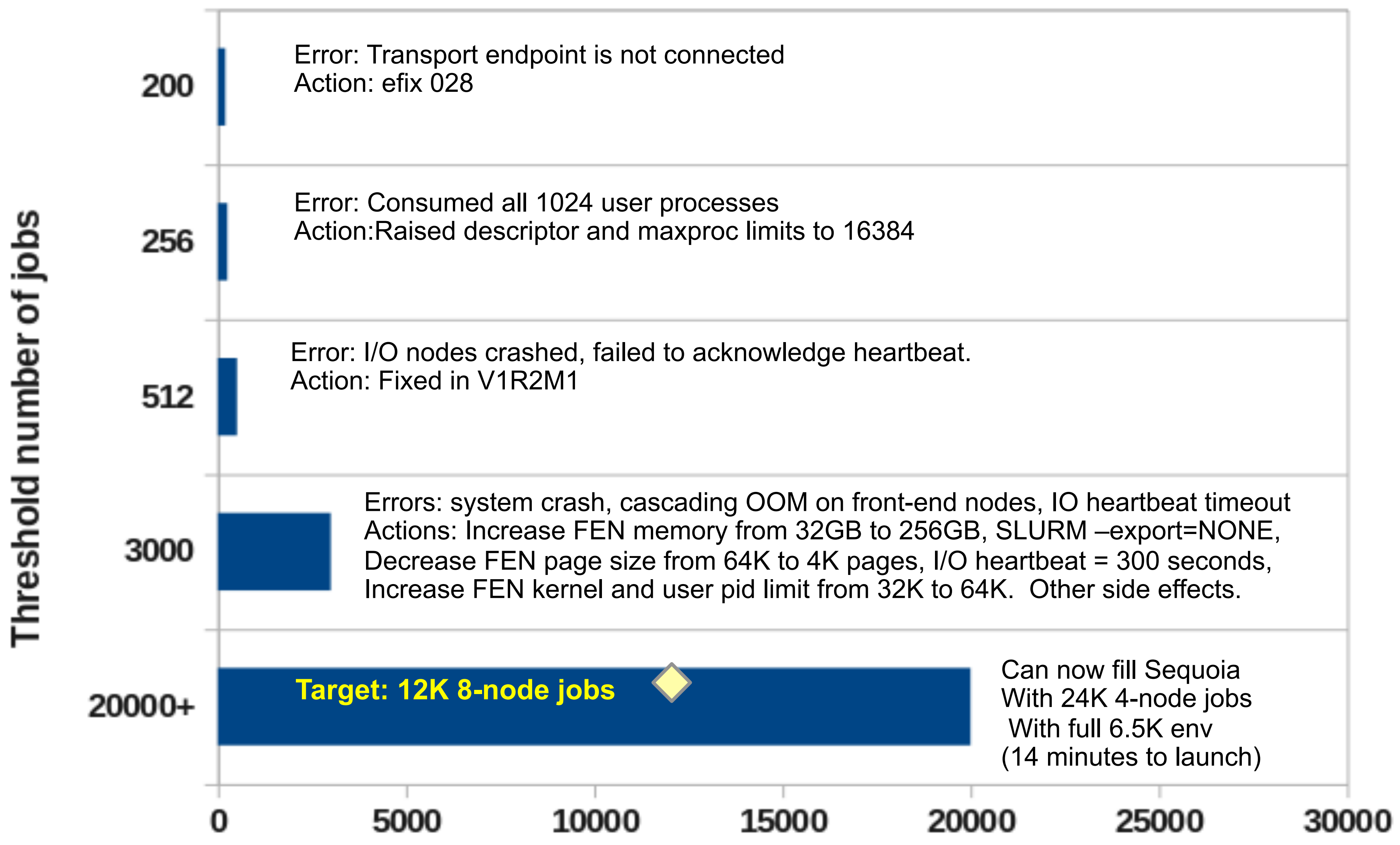
- Socket Bind/Listen differs from POSIX on Linux
  - Bind doesn’t truly grab socket (to support changing interfaces)
  - Listen on “bound” socket can return EADDRINUSE
    - Another socket is already listening on the same port.
    - Open 1k sockets at once (sockets used by srun), you probably will get this at least once
  - Had to fix slurm and other socket code to deal with ‘rare’ case

- SLURM rare corner cases when packing many small jobs into an allocation
  - Block busy
    - Packing made bad assumption when allocated midplanes assigned to slurm in unexpected order
  - Invalid sub-block corner location
    - Packing small jobs into multiple midplanes failed for unexpected pass-thru configuration

Resolving 3k Runs Limit

- I/O heartbeat timeouts at 3k simultaneous launches
  - 64-task per node launches seemed to trigger at smaller scale
    - Also could take 50 seconds to launch each job at 64 TPN
  - IBM could not reproduce either issue, so IBM could not fix
- Both issues triggered by LLNL's default of 'full env' passing
  - Many LLNL apps need 5-10 env variables set, so default to pass them all
    - Our default env has ~5k characters and 100 env variables
  - Turns out environment variables put huge load on BG/Q control system
  - Efix 23 resolved slow 64 tasks-per-node launches with full env
    - Now same speed as 16 tasks-per-node launches
  - New SLURM option --export=NONE passes only specified env vars
    - UQ pipelines main target for this SLURM option
  - I/O heartbeat set to 300 seconds prevents timeouts will full env variables
    - the jobctl_heartbeat parameter in bg.properties
- UQ users can no longer trivially panic the system via runs
  - Ran 20k+ simultaneous sruns (two batch nodes) with full 6.5k environment
  - Took ~14 minutes to get them all started with full env
  - Can now fill Sequoia's 96k nodes with 4 node jobs if needed

Summary: Progression of Concurrent Job Thresholds



Error: Transport endpoint is not connected
Action: efix 028

Error: Consumed all 1024 user processes
Action:Raised descriptor and maxproc limits to 16384

Error: I/O nodes crashed, failed to acknowledge heartbeat.
Action: Fixed in V1R2M1

Errors: system crash, cascading OOM on front-end nodes, IO heartbeat timeout
Actions: Increase FEN memory from 32GB to 256GB, SLURM –export=NONE,
Decrease FEN page size from 64K to 4K pages, I/O heartbeat = 300 seconds,
Increase FEN kernel and user pid limit from 32K to 64K. Other side effects.

Can now fill Sequoia
With 24K 4-node jobs
With full 6.5K env
(14 minutes to launch)

Origins of the CRAM Library

- **Major codes use large regression test suites**
  - Run on every platform daily, often after each commit to the code.
  - A large number of small and short jobs
    - Exercise key code functionality
    - One code's regression text has > 1000 tests
      - ~700 use 1 task, and ~200 use 4 tasks

- **BG/Q users want to run a lot of jobs simultaneously**
  - A2 core is slow, and small tests run slowly
  - BG/Q's sub-block job minimum size is 1 node
    - Have to use 1k nodes for regression test?!
    - Running > 200 regression jobs caused problems!

- **Asked IBM, any way we can pack a node with small jobs?**
  - Answer: **NO**

Origins of the CRAM Library

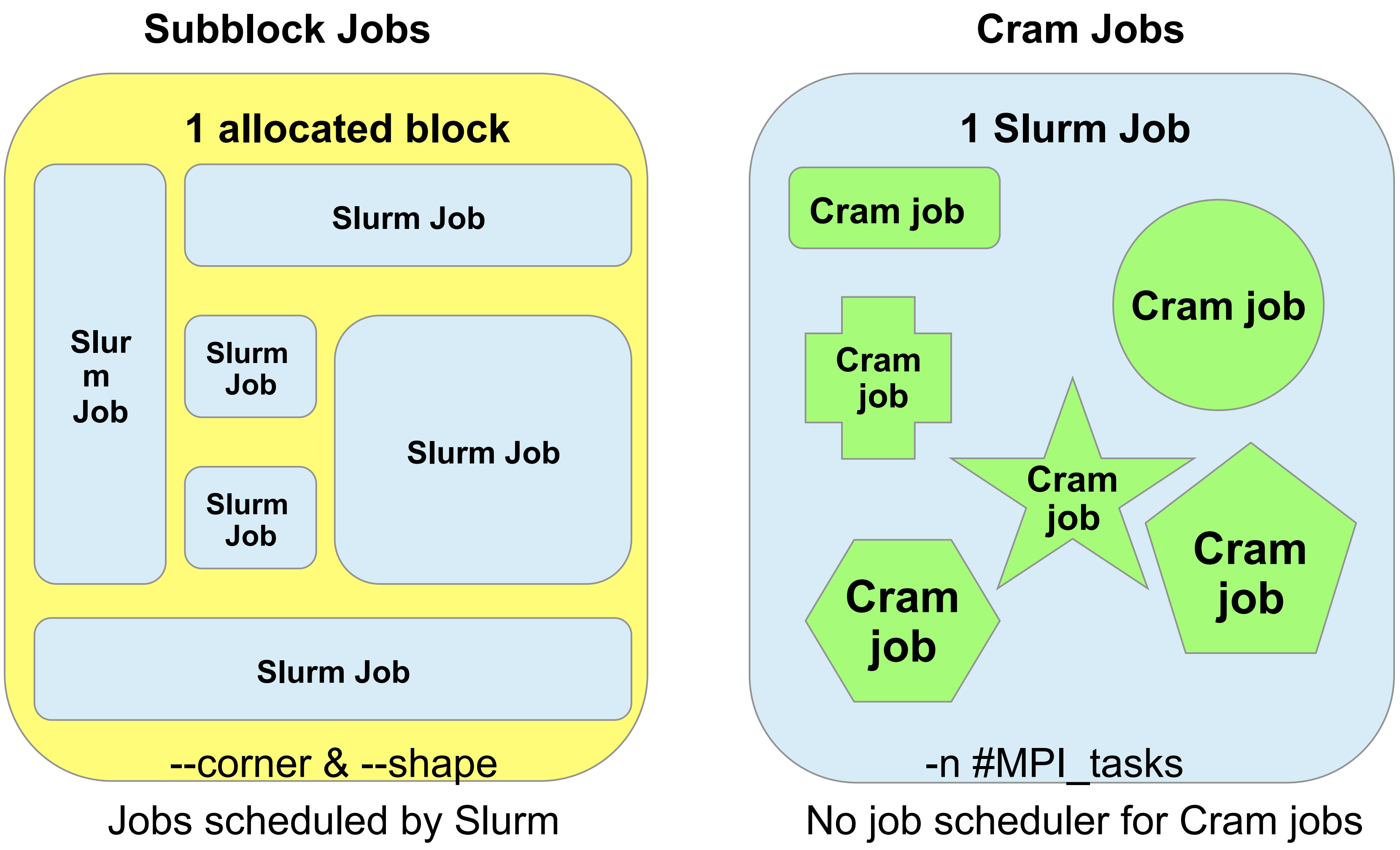
- **New library proposed for running many jobs**
  - Initially named **CLOWNCAR**
  - Heavily leverage off of MPI wrapping tool by Todd Gamblin
  - Design goal: simple to use with no application code changes, just a preload/relink
  - Every job can have unique working directory, args, and env variables

- **Strong community interest but some community objections to name**
  - Renamed to the **CRAM** library
    - Jokingly defined as: Clowncar Renamed to Appease Management
  - Became possible solution to UQ scaling issues also
    - Cram a small number of jobs (e.g. 16) into bigger jobs
    - Cram all jobs in one big allocation

The CRAM Library: Enabling 1+ million simultaneous jobs

- **PMPI-layer library runs many jobs transparently on sub-communicators**
  - CRAM splits MPI_COMM_WORLD into per-job communicators
  - Fools each process into thinking it is running on MPI_COMM_WORLD
- **Limitations:**
  - Application must parse args and environment AFTER MPI_Init
  - No running a new sub-job after one sub-job completes (requires big app changes)
- **Tightly packs jobs, even non-powers of two MPI tasks (unlike BG/Q)**
  - Can pack 64 1 task jobs onto one node!
  - Perfect for regression tests with small task count (700 1 tasks jobs -> 44 nodes at 16TPN)
- **Works with LLNL's tool-killing apps and highly scalable**
  - Initial BG/Q implementation available, contact: Todd Gamblin <gamblin2@llnl.gov>

Subblock Jobs vs. Cram Jobs



Block
Slurm Job
MPI Communicator

Cram has a python interface for rapid creation of job lists

- **Cram command line usage is relatively simple:**

```
cram pack -f cram.job -n 1 my_mpi_application input.1.txt
cram pack -f cram.job -n 1 my_mpi_application input.2.txt
...
cram pack -f cram.job -n 1 my_mpi_application input.1048576.txt

env CRAM_FILE=/path/to/cram.job srun -n 1048576 my_mpi_application
```

- **However, time to create the cram.job file is a bottleneck for 1.5M jobs**
  - Invoking the cram python script 1.5M times: 8 Python launches per second -> 54 hours
  - Best case if invoking a C program 1.5M times: 700 simplest C programs per second -> 37 minutes
- **Need to use CRAM's Python interface when creating millions of jobs**
  - Single python script to generate job list -> 4 minutes

```
cf = CramFile('cram.job', 'w')
for i in xrange(1048576):
    env["SCRATCH_DIR"] = "/p/lscratcha/%s/scratch-%08d" % (user, i)
    args = ["input.%08d" % i]
    cf.pack(1, '/home/%s/ensemble/run-%08d' % (user, i), args, env)
cf.close()
```

We have scaled CRAM to 1.5 million jobs

- We ran 1,572,864 jobs in one 96k node Sequoia allocation

Reading job file, Broadcasting and setting up jobs, Calling MPI_Comm_split	104.5 s
Creating 2 files for each job's stdio/stderr on Lustre	1167.7 s (19 m)
Minimal Memory Impact	< 10MB

- File creation is slow:
  - Need to amortize 19 minutes with large throughput.
- Jobs all run successfully